**摩尔斯电码**

　 摩尔斯电码（又译为摩斯电码）是一种时通时断的信号代码，这种信号代码通过不同的排列顺序来表达不同的英文字母、数字和标点符号等。它由美国人艾尔菲德·维尔发明。

艾尔菲德·维尔构思了一个方案，通过点、划和中间的停顿，可以让每个字元和标点符号彼此独立地发送出去。

作为一种信息编码标准，摩尔斯电码拥有其他编码方案无法超越的长久的生命。摩尔斯电码在海事通讯中被作为国际标准一直使用到1999年。1997年，当法国海军停止使用摩尔斯电码时，发送的最后一条消息是：“所有人注意，这是我们在永远沉寂之前最后的一声呐喊！”

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **字符** | **电码符号** | **字符** | **电码符号** | **字符** | **电码符号** | **字符** | **电码符号** |
| **A** | ．━ | **B** | ━ ．．． | **C** | ━ ．━ ． | **D** | ━ ．． |
| **E** | ． | **F** | ．．━ ． | **G** | ━ ━ ． | **H** | ．．．． |
| **I** | ．． | **J** | ．━ ━ ━ | **K** | ━ ．━ | **L** | ．━ ．． |
| **M** | ━ ━ | **N** | ━ ． | **O** | ━ ━ ━ | **P** | ．━ ━ ． |
| **Q** | ━ ━ ．━ | **R** | ．━ ． | **S** | ．．． | **T** | ━ |
| **U** | ．．━ | **V** | ．．．━ | **W** | ．━ ━ | **X** | ━ ．．━ |
| **Y** | ━ ．━ ━ | **Z** | ━ ━ ．． |  |  |  |  |

**数字**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **字符** | **电码符号** | **字符** | **电码符号** | **字符** | **电码符号** | **字符** | **电码符号** |
| **0** | ━ ━ ━ ━ ━ | **1** | ．━ ━ ━ ━ | **2** | ．．━ ━ ━ | **3** | ．．．━ ━ |
| **4** | ．．．．━ | **5** | ．．．．． | **6** | ━ ．．．． | **7** | ━ ━ ．．． |
| **8** | ━ ━ ━ ．． | **9** | ━ ━ ━ ━ |  |  |  |  |

**ASCII**

美国信息交换标准码(ASCII:American Standard Code for Information Interchange)起始于50年代后期，并最终在1967年定案。最终的代码有26个小写字母，26个大写字母，10个数字，32个符号， 33个控制代码和一个空格。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bin | Dec | Hex | 缩写/字符 | 解释 |
| 00000000 | 0 | 00 | NUL(null) | 空字符 |
| 00000001 | 1 | 01 | SOH(start of headling) | 标题开始 |
| 00000010 | 2 | 02 | STX (start of text) | 正文开始 |
| 00000011 | 3 | 03 | ETX (end of text) | 正文结束 |
| 00000100 | 4 | 04 | EOT (end of transmission) | 传输结束 |
| 00000101 | 5 | 05 | ENQ (enquiry) | 请求 |
| 00000110 | 6 | 06 | ACK (acknowledge) | 收到通知 |
| 00000111 | 7 | 07 | BEL (bell) | 响铃 |
| 00001000 | 8 | 08 | BS (backspace) | 退格 |
| 00001001 | 9 | 09 | HT (horizontal tab) | 水平制表符 |
| 00001010 | 10 | 0A | LF (NL line feed, new line) | 换行键 |
| 00001011 | 11 | 0B | VT (vertical tab) | 垂直制表符 |
| 00001100 | 12 | 0C | FF (NP form feed, new page) | 换页键 |
| 00001101 | 13 | 0D | CR (carriage return) | 回车键 |
| 00001110 | 14 | 0E | SO (shift out) | 不用切换 |
| 00001111 | 15 | 0F | SI (shift in) | 启用切换 |
| 00010000 | 16 | 10 | DLE (data link escape) | 数据链路转义 |
| 00010001 | 17 | 11 | DC1 (device control 1) | 设备控制1 |
| 00010010 | 18 | 12 | DC2 (device control 2) | 设备控制2 |
| 00010011 | 19 | 13 | DC3 (device control 3) | 设备控制3 |
| 00010100 | 20 | 14 | DC4 (device control 4) | 设备控制4 |
| 00010101 | 21 | 15 | NAK (negative acknowledge) | 拒绝接收 |
| 00010110 | 22 | 16 | SYN (synchronous idle) | 同步空闲 |
| 00010111 | 23 | 17 | ETB (end of trans. block) | 传输块结束 |
| 00011000 | 24 | 18 | CAN (cancel) | 取消 |
| 00011001 | 25 | 19 | EM (end of medium) | 介质中断 |
| 00011010 | 26 | 1A | SUB (substitute) | 替补 |
| 00011011 | 27 | 1B | ESC (escape) | 溢出 |
| 00011100 | 28 | 1C | FS (file separator) | 文件分割符 |
| 00011101 | 29 | 1D | GS (group separator) | 分组符 |
| 00011110 | 30 | 1E | RS (record separator) | 记录分离符 |
| 00011111 | 31 | 1F | US (unit separator) | 单元分隔符 |
| 00100000 | 32 | 20 | (space) | 空格 |
| 00100001 | 33 | 21 | ! |  |
| 00100010 | 34 | 22 | " |  |
| 00100011 | 35 | 23 | # |  |
| 00100100 | 36 | 24 | $ |  |
| 00100101 | 37 | 25 | % |  |
| 00100110 | 38 | 26 | & |  |
| 00100111 | 39 | 27 | ' |  |
| 00101000 | 40 | 28 | ( |  |
| 00101001 | 41 | 29 | ) |  |
| 00101010 | 42 | 2A | \* |  |
| 00101011 | 43 | 2B | + |  |
| 00101100 | 44 | 2C | , |  |
| 00101101 | 45 | 2D | - |  |
| 00101110 | 46 | 2E | . |  |
| 00101111 | 47 | 2F | / |  |
| 00110000 | 48 | 30 | 0 |  |
| 00110001 | 49 | 31 | 1 |  |
| 00110010 | 50 | 32 | 2 |  |
| 00110011 | 51 | 33 | 3 |  |
| 00110100 | 52 | 34 | 4 |  |
| 00110101 | 53 | 35 | 5 |  |
| 00110110 | 54 | 36 | 6 |  |
| 00110111 | 55 | 37 | 7 |  |
| 00111000 | 56 | 38 | 8 |  |
| 00111001 | 57 | 39 | 9 |  |
| 00111010 | 58 | 3A | : |  |
| 00111011 | 59 | 3B | ; |  |
| 00111100 | 60 | 3C | < |  |
| 00111101 | 61 | 3D | = |  |
| 00111110 | 62 | 3E | > |  |
| 00111111 | 63 | 3F | ? |  |
| 01000000 | 64 | 40 | @ |  |
| 01000001 | 65 | 41 | A |  |
| 01000010 | 66 | 42 | B |  |
| 01000011 | 67 | 43 | C |  |
| 01000100 | 68 | 44 | D |  |
| 01000101 | 69 | 45 | E |  |
| 01000110 | 70 | 46 | F |  |
| 01000111 | 71 | 47 | G |  |
| 01001000 | 72 | 48 | H |  |
| 01001001 | 73 | 49 | I |  |
| 01001010 | 74 | 4A | J |  |
| 01001011 | 75 | 4B | K |  |
| 01001100 | 76 | 4C | L |  |
| 01001101 | 77 | 4D | M |  |
| 01001110 | 78 | 4E | N |  |
| 01001111 | 79 | 4F | O |  |
| 01010000 | 80 | 50 | P |  |
| 01010001 | 81 | 51 | Q |  |
| 01010010 | 82 | 52 | R |  |
| 01010011 | 83 | 53 | S |  |
| 01010100 | 84 | 54 | T |  |
| 01010101 | 85 | 55 | U |  |
| 01010110 | 86 | 56 | V |  |
| 01010111 | 87 | 57 | W |  |
| 01011000 | 88 | 58 | X |  |
| 01011001 | 89 | 59 | Y |  |
| 01011010 | 90 | 5A | Z |  |
| 01011011 | 91 | 5B | [ |  |
| 01011100 | 92 | 5C | \ |  |
| 01011101 | 93 | 5D | ] |  |
| 01011110 | 94 | 5E | ^ |  |
| 01011111 | 95 | 5F | \_ |  |
| 01100000 | 96 | 60 | ` |  |
| 01100001 | 97 | 61 | a |  |
| 01100010 | 98 | 62 | b |  |
| 01100011 | 99 | 63 | c |  |
| 01100100 | 100 | 64 | d |  |
| 01100101 | 101 | 65 | e |  |
| 01100110 | 102 | 66 | f |  |
| 01100111 | 103 | 67 | g |  |
| 01101000 | 104 | 68 | h |  |
| 01101001 | 105 | 69 | i |  |
| 01101010 | 106 | 6A | j |  |
| 01101011 | 107 | 6B | k |  |
| 01101100 | 108 | 6C | l |  |
| 01101101 | 109 | 6D | m |  |
| 01101110 | 110 | 6E | n |  |
| 01101111 | 111 | 6F | o |  |
| 01110000 | 112 | 70 | p |  |
| 01110001 | 113 | 71 | q |  |
| 01110010 | 114 | 72 | r |  |
| 01110011 | 115 | 73 | s |  |
| 01110100 | 116 | 74 | t |  |
| 01110101 | 117 | 75 | u |  |
| 01110110 | 118 | 76 | v |  |
| 01110111 | 119 | 77 | w |  |
| 01111000 | 120 | 78 | x |  |
| 01111001 | 121 | 79 | y |  |
| 01111010 | 122 | 7A | z |  |
| 01111011 | 123 | 7B | { |  |
| 01111100 | 124 | 7C | | |  |
| 01111101 | 125 | 7D | } |  |
| 01111110 | 126 | 7E | ~ |  |
| 01111111 | 127 | 7F | DEL (delete) | 删除 |

ASCII是一个真正的美国标准，所以他不能很好的满足其他将英语的国家的需要。例如英国的英镑符号(￡)就没有。

**解决方案为代码页。**

在小型机开发初期，就已经严格建立了8位字节。因此，如果使用一个字节来保存字符，则可以由128个附加的字符来补充。

最低的128个代码总是相同的，较高的128个代码则取决于定义代码页的语言。如果用户为PC键盘，显示卡，和打印机指定了一个代码页，然后在PC上创建、编辑和打印文档，一切都很正常，每件事都会保持一致。然而，如果用户试图与使用不同代码页的用户交换文件，就会产生问题。当然，应用程序可以通过将代码页信息与文档一起保存的方式来解决问题，但是且慢，更糟的事情还在后头。

在中国、日本和韩国的象形文字符号大约有21000个，如何容纳这些语言而仍保持和ASCII的某种兼容性呢。

**解决方案为双字节字符集**

双字节字符集（DBCS:double-byte character set）与其他代码页一样，最初的128个代码是ASCII，较高的128个代码中的某些总是跟随者第二个字节（称作首字节和跟随字节）。这两个字节一起定义一个字符，通常是一个复杂的象形文字。

DBCS的问题在于不是每个字符都由两个字节代表，一些字符由一个字节表示，而另一些字符则由两个字符表示。这会引起附加的编程问题。例如，字符串中的字符数不能由字符串的字节数决定。必须剖析字符串来决定其长度，而且必须检查每个字节以确定它是否为双字节字符。

令人惊讶的是这套机制，虽然对付，但仍被程序员所接受了，今天我们使用的C运行库函数都是在这套机制下编写的。

**Unicode 解决方案**

我们面临的问题是世界上的书写语言不能简单地用256个8位代码表示。怎么办？用16位表示呗。Unicode就是这样一个字符集，它的每个字符都是16位宽，而且最大的好处是，您将只需要操作一个字符集，因为Unicode字符集，涵盖了所有文字符号。再也不用考虑代码页之间的转换问题了。

当然，Unicode也有缺点，那就是他的字符串占用的内存是ASCII的两倍。

**char与 wchar\_t**

即便使用Unicode char数据类型仍然表示1个字节的存储空间

如果想定义一个两个字节宽度的字符存储空间 需要使用wchar\_t;

例如

char c = ‘A’;

wchar\_t c = L’A’;

注意紧挨的大写字母L,它将告诉编译器该字符按宽字符保存 – 即每个字符占用2个字节

char \* p = “Hello”;

wchar\_t \* p = L”Hello”;

**世界正在改变**

我们现在尝试着获取字符串的长度

char \* pc = “Hello!”;

iLength = strlen(pc);

字符串长度为6。

wchar\_t \* pw = L”Hello!”;

再次调用strlen

iLength = strlen(pw);

您会发现，iLength 的值为1。为什么？

字符串L”Hello!” 在内存中的格式为。

48 00 65 00 6C 00 6C 00 6F 00 21 00

strlen 会把第一个字节作为字符开始计数，但接着下一个字节是0，则表示字符串结束。

你知道这意味这什么么？如果想支持Unicode那就得重写所有的C运行库函数，当然也没有那么夸张，只要重写所有跟字符串有关的函数就可以了，而且好消息是，这些工作已经做完了。

想要获取一个宽字符的字节数只要调用wcslen就可以了

iLength = wcslen(pw);

**两套字符集 一套维护代码**

Unicode 最大的缺点是程序中的每个字符串都将占用两倍的存储空间。

而且某些地区可能值支持ASCII并不支持Unicode (非常少见。。。)

所以也许您希望建立两个版本的程序---一个处理ASCII字符串，另一个处理Unicode字符串。

虽然这是一个小问题，但由于运行库函数有不同的名称，您也要用不同的方式（char wchar\_t）定义字符，而且宽字符字符串前面还需要加L。

解决办法是------宏

首先是字符串前面的L,我们可以使用TEXT()宏来为我们解决这个问题。

#define TEXT(x) L##x

#define TEXT(x) x

## 成为粘贴符号

不同的运行库函数名称，也可以通过这个方式解决